

where AP is the average pitch angle (laser beams drawing cones), DP is the phase, BP is the magnitude of a tilted plane and a is the rotational angle to the target. Accordingly, the true height of each target is expressed by the equation:

$$\text{True height} = H + \sin(\text{Pitch angle})$$

where H is the observed height for the target. The coefficients generated by this calculation are also stored in memory.

Remarks:

Entrance of the foregoing corrective amendment to the specification is solicited. No new matter has been added as a result of this amendment. The correct formula appears in several source code files forming a part of the specification as originally filed. For example, a printout of a file named CALCTGTS.C is attached. Comments on line 63 recite "Compute the pitch component using the raw angle and range to the target". A "CalculatePitchComponent" subroutine call at line 64 initiates a pitch component calculation and passes an angle and a range to the "CalculatePitchComponent" subroutine. The "CalculatePitchComponent" subroutine begins on line 285 and includes a " $z = \text{Range} * \sin(\text{PitchAngle})$ " formula. The "CalculatePitchComponent" subroutine returns "z" as the pitch component at line 299.

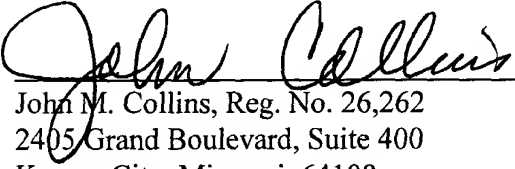
At line 170, a height is calculated with a "CalculateHeight" subroutine call. The "CalculateHeight" subroutine begins at line 366 and includes a " $\text{height} += \text{pitchComp}$ " formula at line 371, where the previously calculated pitch component is added to the height. The "CalculateHeight" subroutine returns the height at line 375.

Accordingly, the present amendment merely conforms the specification on page 17 with the source code. Therefore, it is requested that the amendment be entered prior to examination on the merits.

Any additional fee which is due in connection with this amendment should be applied against our Deposit Account No. 19-0522.

Respectfully submitted,

HOVEY, WILLIAMS, TIMMONS, & COLLINS

By: 
John M. Collins, Reg. No. 26,262
2405 Grand Boulevard, Suite 400
Kansas City, Missouri 64108
(816) 474-9050

ATTORNEYS FOR APPLICANT

(Docket No. 30688)

Application Serial No.: 09/727,632

VERSION WITH MARKINGS TO SHOW CHANGES MADE

Please amend the section entitled Detailed Description of the Preferred Embodiments, page 17 lines 29 and 30, as follows:

$$\text{True height} = H + \underline{R} * \sin(\text{Pitch angle})$$

where H is the observed height for the target and R is the range (distance from hub) to the target. The coefficients generated by this calculation are also stored in memory.

```

5  #include "types.h"
   #include "Chiefeye.h"

10 #define SQU(a)      ((a)*(a))           // Square of a number
   #define DIST(a,b)  sqrt(SQU(a)+SQU(b)) // Distance between two points

15 // process all targets and return number of valid targets found
   // Processing consists of first applying any correction to the measured
   // hub angles by calling RawToReal()
   // Next, the target is triangulated and X,Y determined RealToXY()
   // Finally, the height is calculated and corrected for pitch/cone
   // See Flow Chart dated 3/18/00

20 /* FaceAngle:
   A target parallel to the X axis with it's face toward increasing Y
   has a face angle of 0. The face angle increases as the target is
   rotated counter-clockwise. Since we scan from right-to-left (CCW),
   the face angle can be calculated as:
   FA = atan( (Ygh - Yab) / (Xgh - Xab) )
   */

25 int CalculateTargets()
   {
   int n=0;
   int i, hub;
   TGT* pTgt;
   double Deltax, Deltay;
   double ThicknessOver2;

30   for (i=1; i<=LAST_ID; i++) // process each target
   {
       pTgt=&TargetArray[i];

35       #define HEIGHT(a) pTgt->raw_height[a]
       #define HEIGHT(a) pTgt->raw_height[a]
       if (pTgt->pTgtDescr==NULL)
40         continue; // no target description

       ThicknessOver2 = pTgt->pTgtDescr->TgtThickness/2.0; // needed later

45       // calculate real angles
       for (hub=0; hub<4; hub++)
       {
           if (!pTgt->seen_by[hub])
50             continue; // not seen by this hub

           {
               int page;

```



```

double angle;

// Estimate the Real Angle and determine which coefficients to use
page = DetermineCoeffPage (pTgt->raw_angle_CTR[hub], hub);
pTgt->CoeffPage[hub] = page;

60 // Compute real angles
    pTgt->real_angle_A[hub] = RawToReal(pTgt->raw_angle_A[hub], page);
    pTgt->real_angle_H[hub] = RawToReal(pTgt->raw_angle_H[hub], page);
    pTgt->real_angle_CF[hub] = RawToReal(pTgt->raw_angle_CF[hub], page);

// Compute the pitch component using the raw angle and range to the target
pTgt->pitch_component[hub] = CalculatePitchComponent (
    pTgt->raw_angle_CTR[hub],
    pTgt->raw_range[hub],
    page );

70 }
    }

    pTgt->computed_height[hub] = HEIGHT(hub); // can be used instead of ratio
}

// Triangulate TOP SCANS
if (pTgt->seen_by[LT] && pTgt->seen_by[RT]) // If seen by both top hubs...
{
    // process top laser scans
    RealToXY (pTgt->real_angle_A[LT], pTgt->real_angle_A[RT],
        pTgt->CoeffPage[LT], pTgt->CoeffPage[RT],
        &pTgt->TopXA, &pTgt->TopYA);

80 RealToXY (pTgt->real_angle_H[LT], pTgt->real_angle_H[RT],
    pTgt->CoeffPage[LT], pTgt->CoeffPage[RT],
    &pTgt->TopXH, &pTgt->TopYH);

// Compute Face Center
pTgt->TopXFaceCtr = (pTgt->TopXA + pTgt->TopXH) / 2.0;
pTgt->TopYFaceCtr = (pTgt->TopYA + pTgt->TopYH) / 2.0;
// and Face Angle
pTgt->TopFaceAngle = atan2(pTgt->TopYH - pTgt->TopYA, pTgt->TopXH - pTgt->TopXA);

90 // Get corrections for translating face x,y to centroid x,y for target thickness
GetXYtranslation(ThicknessOver2, pTgt->TopFaceAngle, &Deltax, &Deltay);

pTgt->XTop = pTgt->TopXFaceCtr + Deltax;
pTgt->YTop = pTgt->TopYFaceCtr + Deltay;

95 //DebugPrintf("TOP XYZ %5.1f %5.1f %5.1f\n", pTgt->XTop, pTgt->YTop, pTgt->ZTop);
}

// Triangulate BOTTOM SCANS

```

```

100     if (pTgt->seen_by[LB] && pTgt->seen_by[RB]) // IF seen by both lower hubs...
    {
        // process bottom laser scans
        RealToXY (pTgt->real_angle_A[LB], pTgt->real_angle_A[RB],
                  pTgt->CoeffPage[LB], pTgt->CoeffPage[RB],
                  &pTgt->BotXA, &pTgt->BotYA);

105     RealToXY (pTgt->real_angle_H[LB], pTgt->real_angle_H[RB],
                  pTgt->CoeffPage[LB], pTgt->CoeffPage[RB],
                  &pTgt->BotXH, &pTgt->BotYH);

110     // Compute Face Center
    pTgt->BotFaceCtr = (pTgt->BotXA + pTgt->BotXH) / 2.0;
    pTgt->BotFaceCtr = (pTgt->BotYA + pTgt->BotYH) / 2.0;
    // and Face Angle
    pTgt->BotFaceAngle = atan2(pTgt->BotYH - pTgt->BotYA, pTgt->BotXH - pTgt->BotXA);

115     // Get corrections for translating face x,y to centroid x,y for target thickness
    GetXYtranslation(ThicknessOver2, pTgt->BotFaceAngle, &Deltax, &Deltay);

    pTgt->XBot = pTgt->BotFaceCtr + Deltax;
    pTgt->YBot = pTgt->BotFaceCtr + Deltay;

120     //DebugPrintf("BOT XYZ %5.1f %5.1f %5.1f\n",pTgt->XBot, pTgt->YBot, pTgt->ZBot);
    }

    // Do height calculations for each hub
    for (hub=0; hub<4; hub++)
    {
125         double x0, y0;
        double ratio;

        if (hub==LT || hub==RT) // Top hub
        {
130             if (!(pTgt->seen_by[LT] && pTgt->seen_by[RT]))
                continue; // not seen by both hubs

            ComputeIntercept (pTgt->real_angle_CF[hub],

135                             pTgt->TopXA, pTgt->TopYA,
                             pTgt->TopXH, pTgt->TopYH,
                             pTgt->CoeffPage[hub],
                             &x0, &y0);

140             ratio = CalculateSpatialRatio ( pTgt->TopXA, pTgt->TopYA,
                                                pTgt->TopXH, pTgt->TopYH,
                                                x0,
145             y0 );
        }
    }
    else // Bottom hub
    {

```

```

150         if (! (pTgt->seen_by[LB] && pTgt->seen_by[RB]))
151             continue; // not seen by both hubs
152
153         ComputeIntercept ( pTgt->real_angle_CF [hub],
154
155             pTgt->BotXA, pTgt->BotYA,
156             pTgt->BotXH, pTgt->BotYH,
157             pTgt->CoeffPage [hub],
158             &x0, &y0 );
159
160         ratio = CalculateSpatialRatio ( pTgt->BotXA, pTgt->BotYA,
161             pTgt->BotXH, pTgt->BotYH,
162             x0, y0 );
163     }
164     pTgt->X0 [hub] = x0; pTgt->Y0 [hub] = y0;
165     pTgt->SpatialRatio [hub] = ratio;
166
167     // now, calculate height of this hub by including all info
168     // We can do it the 'old' way, or we can use Ratio technique
169     if (!USE_RATIO)
170         pTgt->Height [hub] = CalculateHeight ( pTgt->computed_height [hub],
171             1.0,
172             pTgt->pitch_component [hub],
173             pTgt->CoeffPage [hub] );
174     else // calculate using ratio
175         pTgt->Height [hub] = CalculateHeight ( ratio,
176             pTgt->pTgtDescr->RatioMult,
177             pTgt->pitch_component [hub],
178             pTgt->CoeffPage [hub] );
179
180     // DebugPrintf ("ID=%02d, (Page %d), PC=%5.1f, CH=%5.1f, RawH=%5.1f, FinalHeight=%5.1f",
181         //
182         // pTgt->ID, pTgt->CoeffPage [hub],
183         // pTgt->pitch_component [hub],
184         // Coeff [pTgt->CoeffPage [hub]] [H],
185         // ratio*pTgt->pTgtDescr->RatioMult,
186         // pTgt->raw_height [hub],
187         // pTgt->Height [hub] );
188
189     if (pTgt->seen_by[LB] && pTgt->seen_by[LT] && pTgt->seen_by[RB] && pTgt->seen_by[RT])
190     {
191         // Now summarize everything
192         pTgt->X = (pTgt->XTOP + pTgt->XBot) /2;
193         pTgt->Y = (pTgt->YTOP + pTgt->YBot) /2;
194         pTgt->Z = (pTgt->Height [0] + pTgt->Height [1] + pTgt->Height [2] + pTgt->Height [3]) /4;
195         printf ("Z %d %f %f %f\n", pTgt->ID, pTgt->computed_height [LB], pTgt->computed_height [RB], pTgt->
196             >computed_height [LT], pTgt->computed_height [RT] );
197         // Compute averages using exponential filter
198         #define FILTR_CONST // 10.0 is slow, 0.0 does no filtering
199         Cfg.Filter

```

```

195 #define STEP_POINT    10.0 // step immediately if change is more than this value
    // check for big changes - jump directly
    if (fabs(pTgt->XAvg - pTgt->X) > STEP_POINT)    pTgt->XAvg = pTgt->X;
    if (fabs(pTgt->YAvg - pTgt->Y) > STEP_POINT)    pTgt->YAvg = pTgt->Y;
    if (fabs(pTgt->ZAvg - pTgt->Z) > STEP_POINT)    pTgt->ZAvg = pTgt->Z;

200     if (FILTER_CONST > 10.0 || FILTER_CONST < 0.0)    FILTER_CONST = 0.0;    // catch garbage
    pTgt->XAvg = (pTgt->X + FILTER_CONST*pTgt->XAvg) / (FILTER_CONST+1);
    pTgt->YAvg = (pTgt->Y + FILTER_CONST*pTgt->YAvg) / (FILTER_CONST+1);
    pTgt->ZAvg = (pTgt->Z + FILTER_CONST*pTgt->ZAvg) / (FILTER_CONST+1);

205     pTgt->isvalid = 2;
    n++;    // count the target
    }
    }
    return n;
210 }

    double Coeff[8][NBR_COEFF];    // coefficient array

215     int DetermineCoeffPage ( double raw_angle, int hub )
    {
        double angle;

        angle = raw_angle + Coeff[hub][ALL];    // assume front 'page'
        angle = asin(sin(angle));    // normalize between -PI and PI

        if ((hub==LB || hub==LT) && angle<0) ||
            (hub==RB || hub==RT) && angle>0)
225             return (hub);    // positive Y hemisphere
        return (hub+4);    // negative Y hemisphere
230     }

    double RawToReal(double Raw, int Page)
    {
        double Real;
        double main, first, second;
        extern int hCalView;

235         if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
            return Raw; // no correction applied
    }

```


240

```
// Compute using the nifty equation
Real= Raw + Coeff[Page][ALL] +
```

245

```
    Coeff[Page][BLU]*sin(Raw +Coeff[Page][DUL]) +
    Coeff[Page][CLU]*sin(Raw*2+Coeff[Page][ELL]) +
    Coeff[Page][FLU]*sin(Raw*3+Coeff[Page][GLU]) +
    Coeff[Page][HLU]*sin(Raw*4+Coeff[Page][ILU]) +
    0.0;
```

250

```
    return Real;
}
```

255

```
int RealToXY (double Theta1, double Theta2, int Page1, int Page2, double* pX, double* pY)
{
    double x,x1,x2,y,y1,y2;
    double w; // check calculation of Y
    double radius,phase;
    extern int hCalView;
```

260

```
    if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
    {
```

```
        x1 = 0.0;
```

```
        y1 = 0.0;
```

```
        x2 = HubSpan;
```

```
        y2 = 0.0;
```

265

```
    }
    else
```

```
    { // use the info from the coefficient parameters
```

```
        x1 = Coeff[Page1][X];
```

```
        y1 = Coeff[Page1][Y];
```

```
        x2 = Coeff[Page2][X];
```

```
        y2 = Coeff[Page2][Y];
```

270

```
    } // from Jim's reduced equation
```

275

```
    x = (x1*tan(Theta1) - x2*tan(Theta2) - y1 + y2) / (tan(Theta1) - tan(Theta2));
```

```
    // substituting into original equation gives Y
```

```
    y = y1 + (x-x1)*tan(Theta1);
```

```
    w = y2 + (x-x2)*tan(Theta2);
```

```
    *pX = x;
```

```
    *pY = y;
```

```
    return 0;
}
```

280

```
}
```

285

```
double CalculatePitchComponent(double Azimuth, double Range, int page)
{
```

```

double z;
double PitchAngle;

290     if (Cfg.ApplyCorrection == 0) // are we displaying un-corrected readings?
        return 0.0; // don't apply corrections

295     PitchAngle = Coeff[page][AP] + // cone correction
        Coeff[page][BP]*sin(Azimuth + Coeff[page][DP]); // phase dependent tilt

        z = Range * sin(PitchAngle); // tilted plane correction
        // DebugPrintf("Pitch angle %f, height correction %5.1f", PitchAngle, z);

300     return (z);
    }

305     void GetXYtranslation(double ThicknessOver2, double face_angle,
    {
        double *DeltaX, double *DeltaY
        double theta;

310     // The offset from center and thickness are along the X and Y
        // axes, respectively, in a coordinate system XY that is rotated
        // by the face angle from xy. The equations to go from XY to xy are:
        //
        // x = X * cos(theta) - Y * sin(theta)
        // y = X * sin(theta) + Y * cos(theta)

315     // face_angle is rotation of face of target from facing
        // +y direction. For purposes of computing the rotated
        // coordinate system angle, our xy axis are 180 degrees
        // from this.
        theta = face_angle + PI;

20
        *DeltaX = -ThicknessOver2 * sin(theta);
325     *DeltaY = ThicknessOver2 * cos(theta);
    }

330 // This function calculates the point at which a line drawn thru (x1,y1) and (x2,y2)
// intercepts a line from the hub at 'angle' from the x axis.
// I plan to return non-zero if no solution, but currently doesn't check
int ComputeIntercept ( double angle, double x1, double y1, double x2, double y2, int page, double* px0,
double* py0)
{

```

```

335     double x,y;
        double yCheck;
        double TanAngle;
        double xHub, yHub;

340         TanAngle = tan(angle);
        xHub = Coeff[page][X];
        yHub = Coeff[page][Y];

345         // From equations generated from slope-intercept solution:
        x = (xHub * TanAngle - yHub + y1 - x1 * (y2-y1)/(x2-x1))/(TanAngle - (y2-y1)/(x2-x1));
        y = yHub + (x - xHub) * TanAngle;
        yCheck = y1 + (x - x1) * (y2-y1)/(x2-x1);           // check calc.

350         *px0 = x;
        *py0 = y;

        return 0;
    }

355     double CalculateSpatialRatio (double x1, double y1, double x2, double y2, double x0, double y0)
    {
        double a,b;
        double ratio;

360         a = DIST ((x0-x1), (y0-y1));
        b = DIST ((x2-x0), (y2-y0));

        ratio = (a-b)/(a+b);
        return (ratio);
    }

365     double CalculateHeight (double ratio, double mult, double pitchComp, int page)
    {
        double height;

370         height = ratio * mult;           // multiply ratio by target slope
        height += pitchComp;              // adjust for pitch error
        height += Coeff[page][H];         // compensate for hub height
        height -= LaserSpan / 2.0;        // translate plane to between laser beams

375     return height;
    }

```